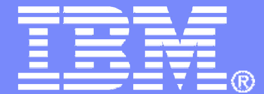


IBM Almaden Research Center

Jaql

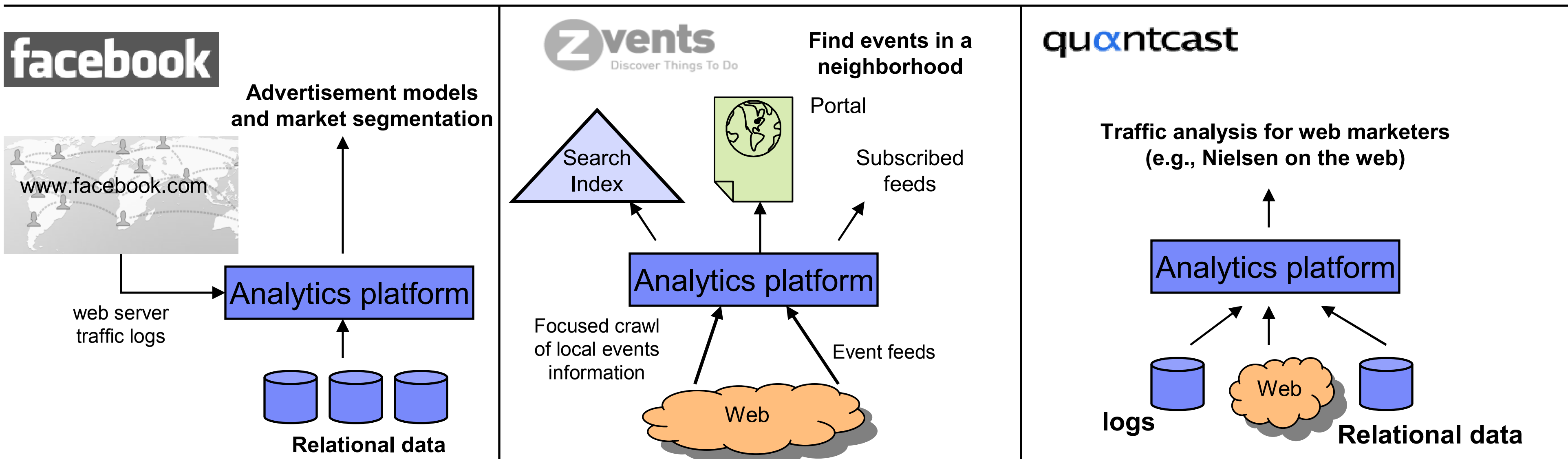
Running Pipes in the Clouds

Kevin Beyer, Vuk Ercegovac, Eugene Shekita,
Jun Rao, Ning Li, Sandeep Tata
IBM Almaden Research Center
http://code.google.com/p/jaql/

© 2009 IBM Corporation

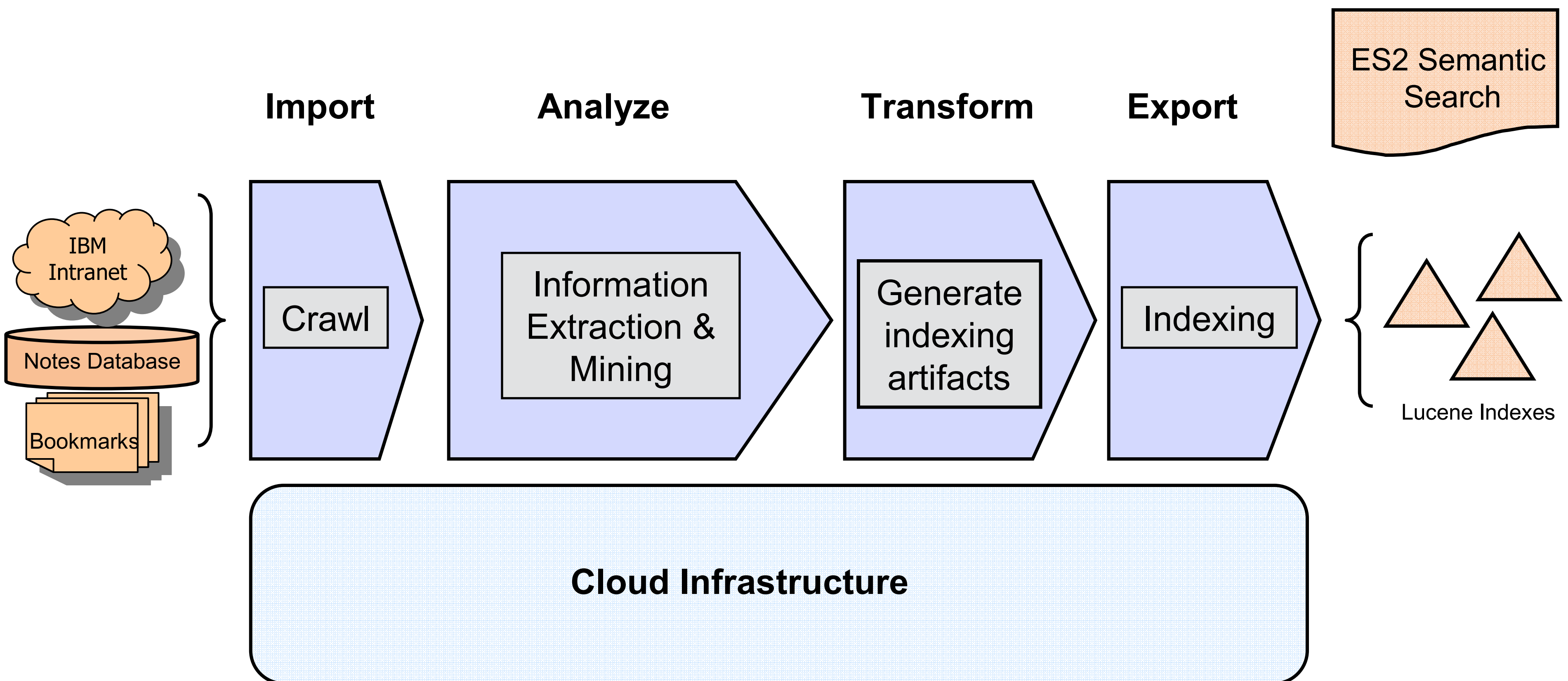
Motivating Scenarios

- Driving forces
  - Data characteristics
    - Unstructured/Semi-structured (e.g., call-center records, pathology reports, ..)
    - Dynamic & Continuously Evolving (e.g., click streams, application logs, system logs)
    - Massive Scale (several terabytes to petabytes ..)
  - Nature of analytics
    - Compute intensive
    - Evolving
      - Demands a **flexible** platform to dynamically accommodate new analytic flows



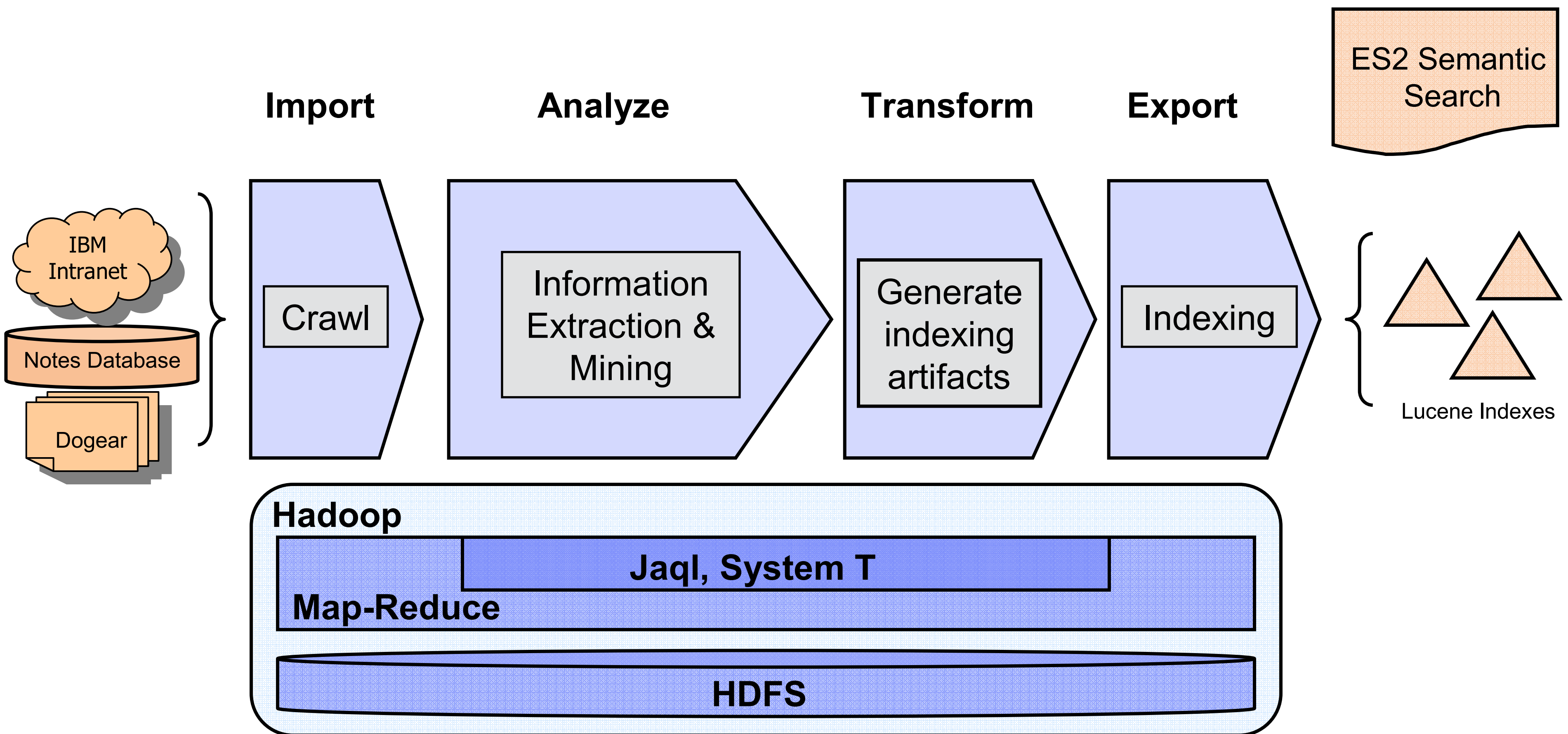
 © 2009 IBM Corporation

ES2: Intranet Data → Semantic Search



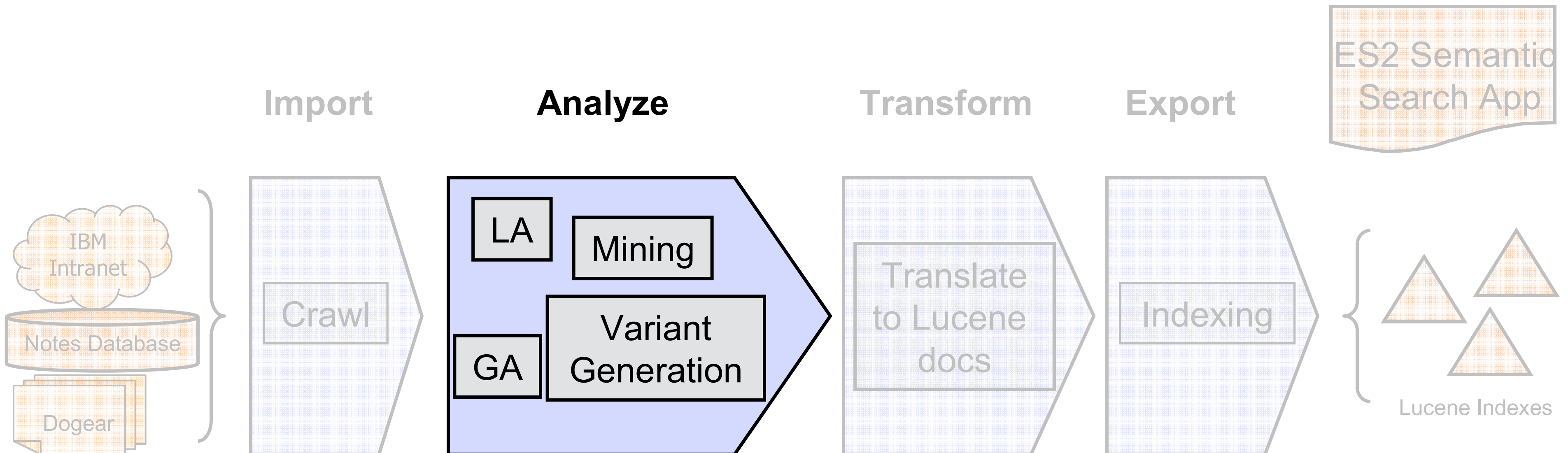
© 2009 IBM Corporation

ES2: Intranet Data → Semantic Search



 © 2009 IBM Corporation

ES2's Analysis Stage



<u>Local Analysis:</u>
System T extracts structured information, e.g., candidate home pages w/ *person* name

<u>Global Analysis:</u>
Jaql finds the best pages within groups of pages, e.g., *personal homepage*

<u>Mining:</u>
Use **map-reduce** for mining
- Acronym extraction
- Geo-classification
- Learning regular expressions

© 2009 IBM Corporation

Goals for Jaql

- Semi-structured data manipulation
  - Use JSON as a data model
    - Data is converted to/from JSON view
    - Most data has a natural JSON representation
      - Relational tables, CSV, XML, …
- Easily extended
  - E.g., Java, Python, JavaScript, …
- Exploit massive parallelism using Hadoop
  - Queries compiled to map-reduce jobs

© 2009 IBM Corporation

JSON Examples

[] == array, {} == record or object, xxx: == field name

Message Log

```
[
 { from:17, to: 19, msg:"…",
      phones: ["+1-415-555-1212", "+1-408-555-1234"], names:["Jill","Jenny"] },
 { from: 17, to: 12, msg:"…", names:["Jack","Jill"]},
 { from: 17, to: 12, msg:"…", phones: ["+1-415-555-2345"] }
]
```

User Info

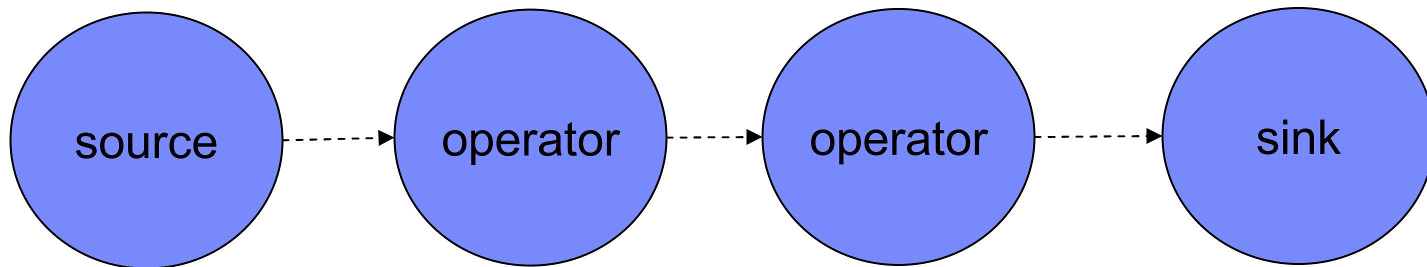
```
[
 { id: 12, name: "Joe Smith",  bday: date("1971-03-07"), zip: 94114 },
 { id: 17, name: "Ann Jones", bday: date("1973-02-04"), zip: 94110 },
 { id: 19, name: "Alicia Fox",  bday: date("1975-04-20"), zip: 94114 }
]
```

City Info

```
[
 { city: "San Francisco", zips: [94110,94112,94114], areacodes: [415] },
 { city: "San Jose", zips: [95120,95123], areacodes: [408] }
]
```

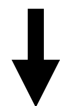

© 2009 IBM Corporation

Writing a pipeline in Jaql



Find users in zip 94114

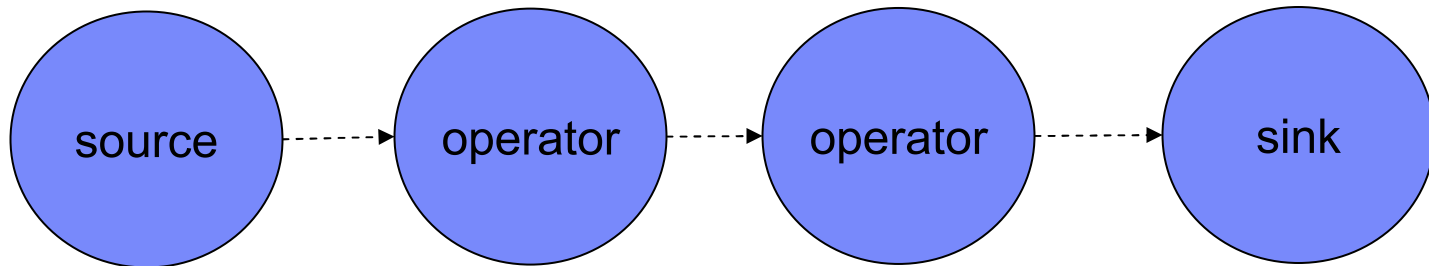
```
[
 { id: 12, name: "Joe Smith",  bday: date("1971-03-07"), zip: 94114 },
 { id: 17, name: "Ann Jones", bday: date("1973-02-04"), zip: 94110 },
 { id: 19, name: "Alicia Fox",  bday: date("1975-04-20"), zip: 94114 }
]
```



```
[
  { id: 12, name: "Joe Smith" },
  { id: 19, name: "Alicia Fox" }
]
```


© 2009 IBM Corporation

Writing a pipeline in Jaql: read



Find users in zip 94114

Query

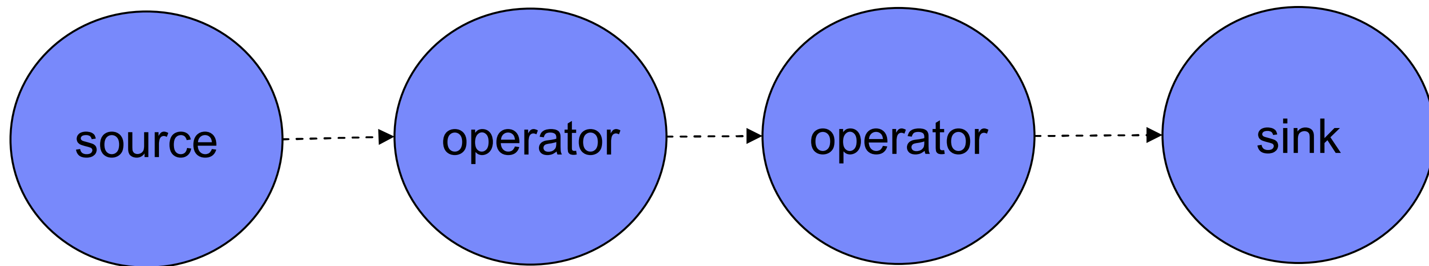
read("hdfs:users");

Data

```
[
 { id: 12, name: "Joe Smith",
      bday: date("1971-03-07"), zip: 94114 },
 { id: 17, name: "Ann Jones",
      bday: date("1973-02-04"), zip: 94110 },
 { id: 19, name: "Alicia Fox",
      bday: date("1975-04-20"), zip: 94114 }
]
```


© 2009 IBM Corporation

Writing a pipeline in Jaql: filter



Find users in zip 94114

Query

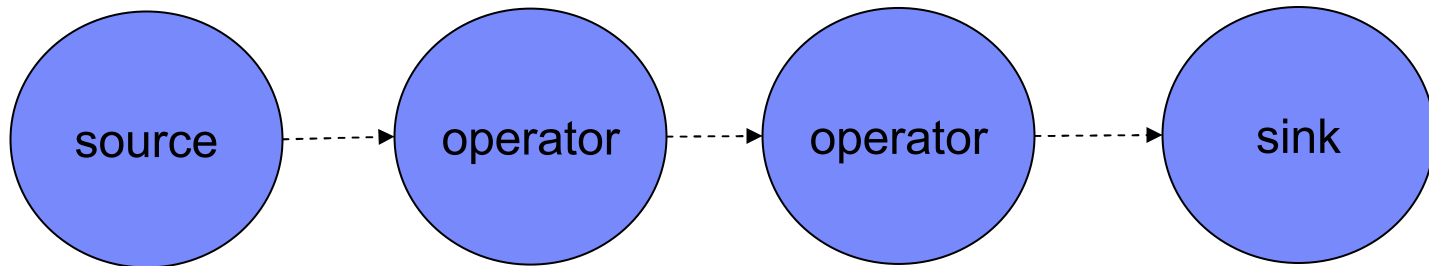
```
read("hdfs:users")
-> filter $.zip == 94114;
```

Data

```
[
 { id: 12, name: "Joe Smith",
     bday: date("1971-03-07"), zip: 94114 },
 { id: 19, name: "Alicia Fox",
     bday: date("1975-04-20"), zip: 94114 }
]
```


© 2009 IBM Corporation

Writing a pipeline in Jaql: transform



Find users in zip 94114

Query

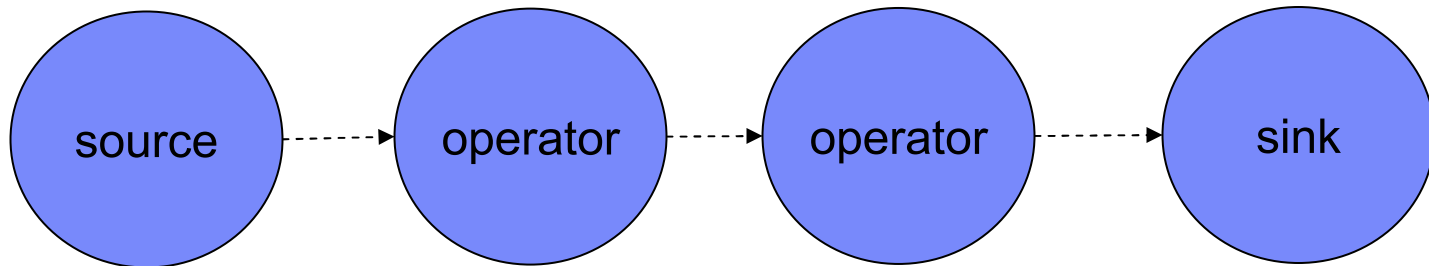
```
read("hdfs:users")
-> filter $.zip == 94114
-> transform { $.id, $.name };
```

Data

```
[
 { id: 12, name: "Joe Smith" },
 { id: 19, name: "Alicia Fox" }
]
```


© 2009 IBM Corporation

Writing a pipeline in Jaql: write



Find users in zip 94114

Query

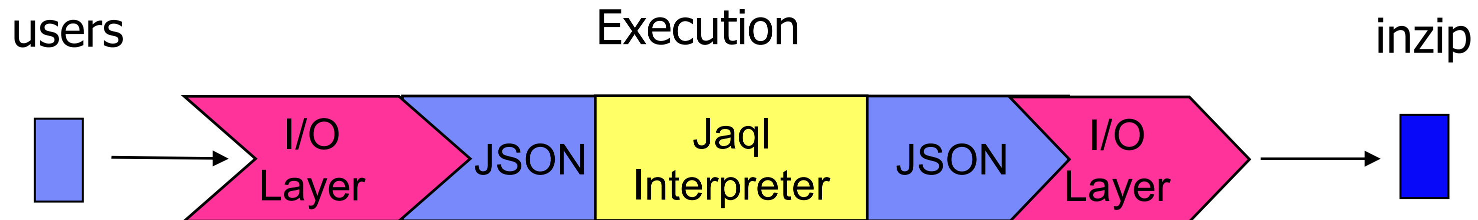
```
read("hdfs:users")
-> filter $.zip == 94114
-> transform { $.id, $.name }
-> write("hdfs:inzip");
```

Data

```
[
 { id: 12, name: "Joe Smith" },
 { id: 19, name: "Alicia Fox" }
]
```


© 2009 IBM Corporation

JAQL I/O

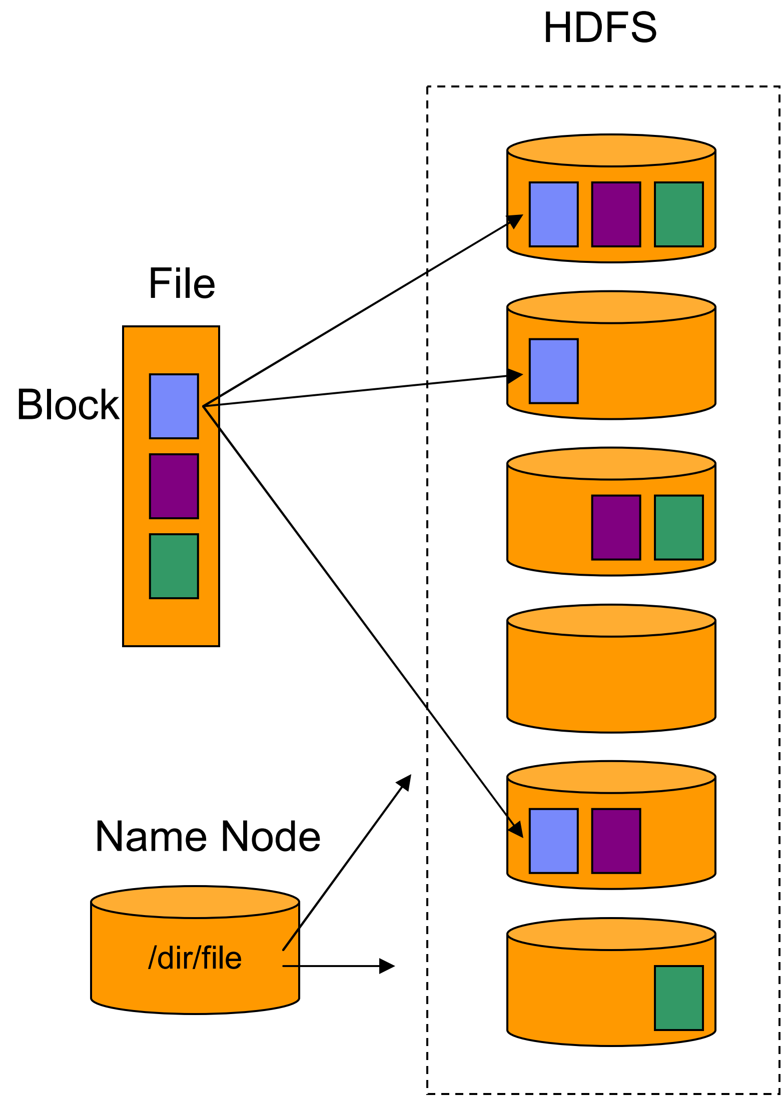


- I/O layer abstracts details of data location + format
  - Examples of data stores:
    - HDFS, HBase, Amazon's S3, local FS, HTTP request, JDBC call
  - Examples of data formats:
    - JSON text, CSV, XML (default is JSON binary)
- JSON used as a data model

© 2009 IBM Corporation

Storing Files in HDFS

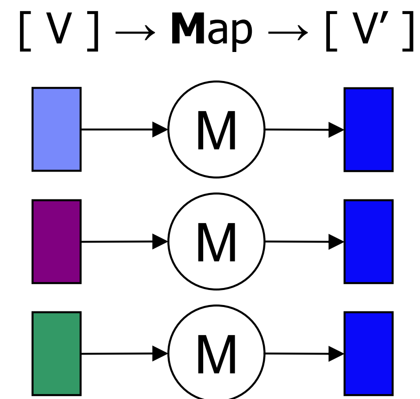
- File is divided into blocks
  - Default block is 64MB
- Blocks are replicated on HDFS nodes
  - Default replication is three times
- The name node tracks file names and block locations



© 2009 IBM Corporation

Map Jobs in Hadoop

- JobConf
  - Specifies job configuration
- InputFormat
  - Abstract data source
  - Creates Splits and RecordReader
- Split
  - Abstract notion of a data chunk
  - Each split handled by a map task
  - Contains location hints
- RecordReader
  - Produces stream of values from a Split
- Map Task
  - Processes values in one Split
  - Calls user map code
  - Reevaluated on error
- OutputFormat
  - Abstract data sink



- Infrastructure handles
  - Parallelism
  - Load balancing
  - Fault-tolerance

© 2009 IBM Corporation

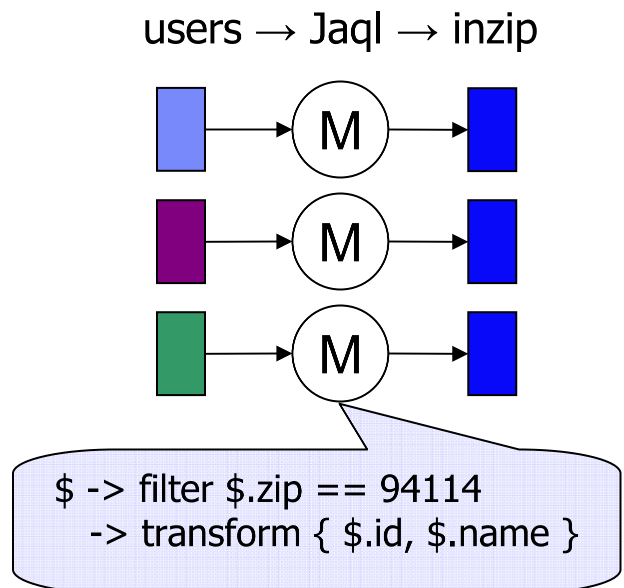
Jaql using Map

```
read("hdfs:users")
-> filter $.zip == 94114
-> transform { $.id, $.name }
-> write("hdfs:inzip");
```

Rewrite Engine

Equivalent map-reduce job in Jaql

```
mapReduce({
input   : {uri: "hdfs:users" },
map     : fn($) ( $ -> filter $.zip == 94114
                    -> transform { $.id, $.name } ),
output  : { uri: "hdfs:inzip" } })
```



© 2009 IBM Corporation

User-defined Functions

Identify phone numbers in each message

```
read("hdfs:rawMsgs")
-> transform {$.*, phones: findPhones($.msg)};
```

Express user code with:
- -Java
- -Inline scripting
- -…

Input data

```
[
 { from:17, to: 19, msg:"…" }
 { from: 17, to: 12, msg:"…" },
 { from: 17, to: 19, msg:"…" }
]
```

Output data

```
[
 { from:17, to: 19, msg:"…",
    phones: ["+1-415-555-1212",
             "+1-408-555-1234"] },
 { from: 17, to: 12, msg:"…"},
 { from: 17, to: 19, msg:"…",
    phones: ["+1-415-555-2345"] }
]
```


© 2009 IBM Corporation

Inline Scripting

```
script python <<END

import re
import phoneUtil

phoneRE = re.compile('[+x]?[0-9\-()]+')

def findPhones(text):
    for p in phoneRE.findall(text):
        p = phoneUtil.standardize(p)
        if p != None:
            yield p

END;

read("hdfs:rawMsgs")
-> transform {$.*, phones: findPhones($.msg)};
```

- Easily integrate with other programming languages
  - Python first
  - then JavaScript, Ruby, Perl, Unix shell, etc

© 2009 IBM Corporation

Grouping

Find the number of times the each sender mentioned an area code

```
read("hdfs:rawMsgs")
-> expand for( $p in findPhones($.msg) ) [{ $.from, area: area($p) }]
-> group by $g = ({$.from, $.area})
     into { $g.*, n: count($) };
```

Input data

```
[
 { from:17, to: 19, msg:"…" }
 { from: 17, to: 12, msg:"…" },
 { from: 17, to: 19, msg:"…" }
]
```

→

After expand

```
[
 { from: 17, area: 415 },
 { from: 17, area: 408 },
 { from: 17, area: 415 }
]
```

→

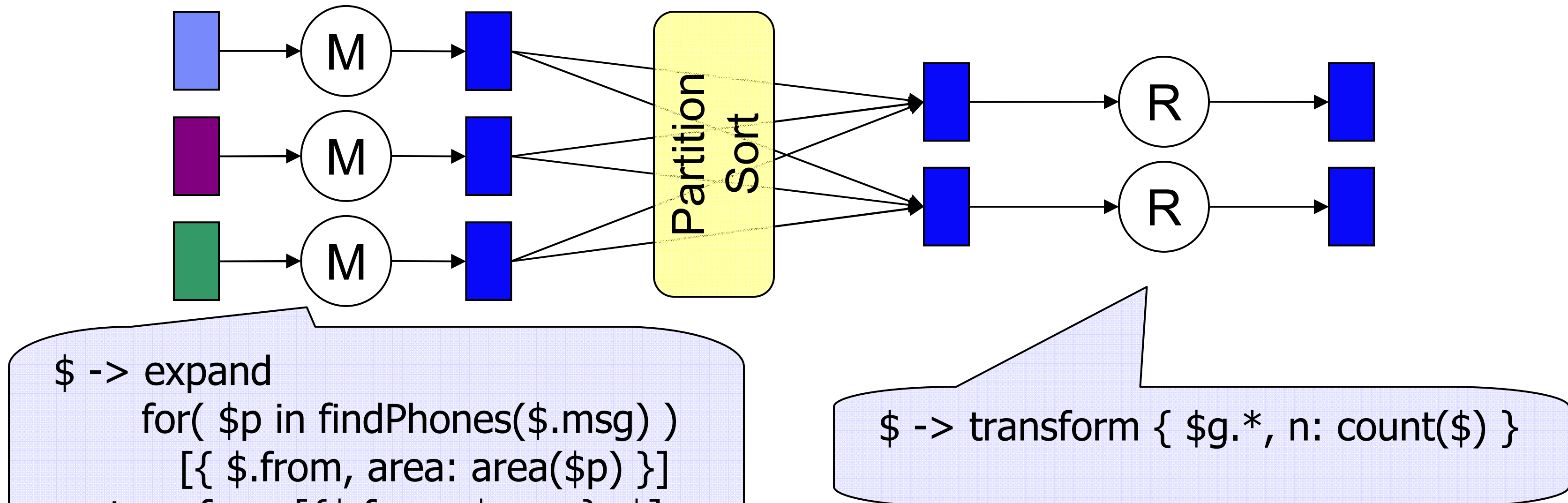
Output data

```
[
 { from: 17, area: 415, n: 2 },
 { from: 17, area: 408, n: 1 }
]
```


© 2009 IBM Corporation

Map-Reduce Jobs in Hadoop

[V] → **M**ap → [K, V′] → Shuffle → [K, [V′]] → Reduce → [V″]

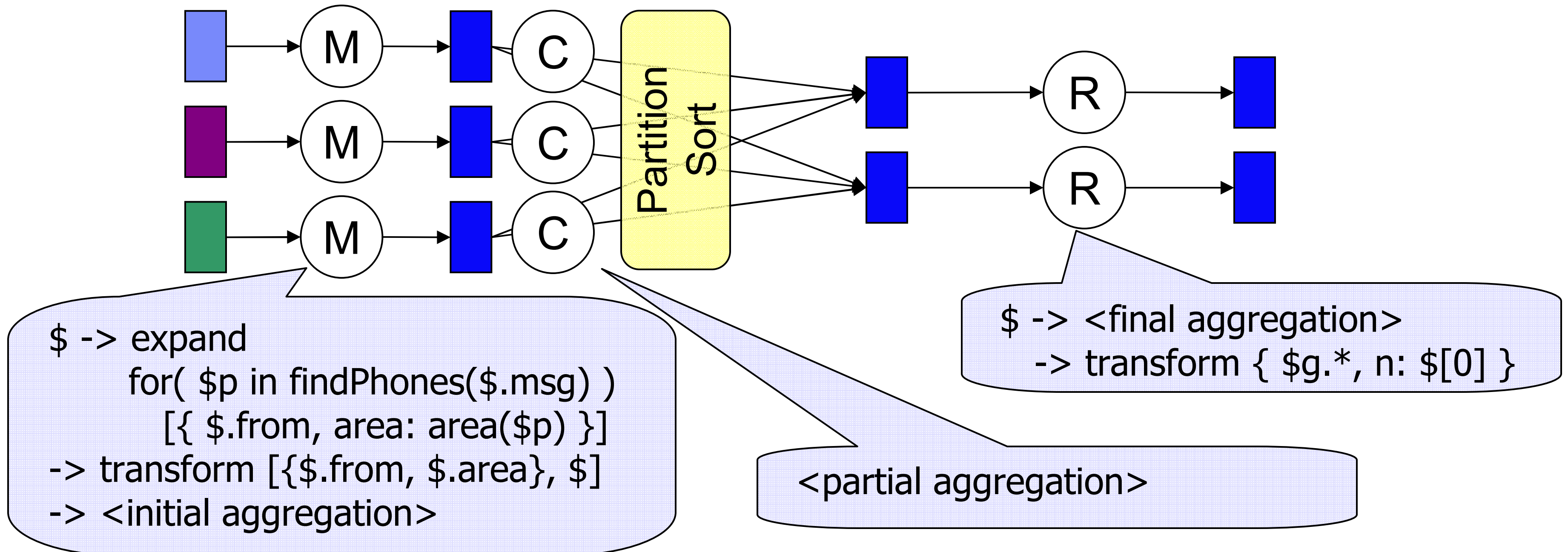



```
mapReduce({
input   : {uri: "hdfs:rawMsgs" },
map     : fn($) ($ -> expand for( $p in findPhones($.msg) )
                          [{ $.from, area: area($p) }]
                     -> transform [{$.from, $.area}, $] ),
reduce : fn($g,$) ($ -> transform { $g.*, n: count($) }),
output  : { uri: "hdfs:out" } })
```

 © 2009 IBM Corporation

Exploiting Combiners with Algebraic Aggregates

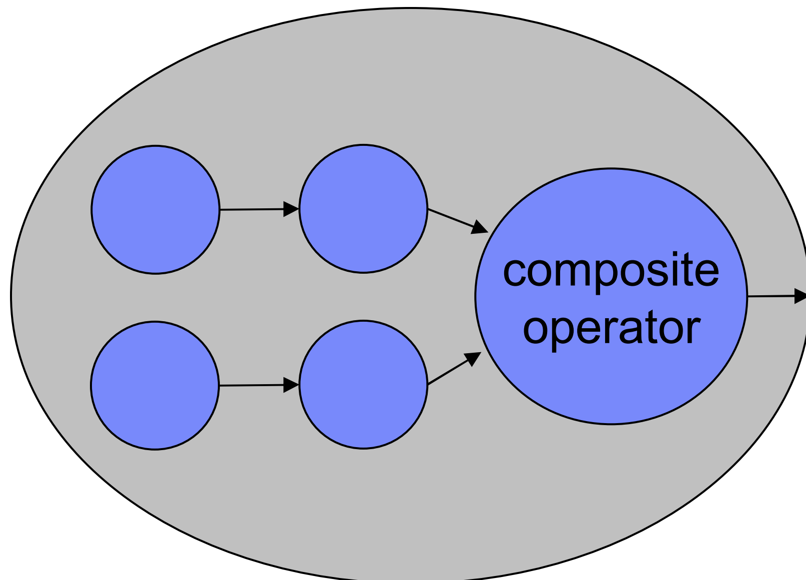
[V] → **M**ap → [K, V′] → Shuffle → [K, [V′]] → Reduce → [V″]




```
mrAggregate({
input   : {uri: “hdfs:rawMsgs” },
map     : fn($) ($ -> expand for( $p in findPhones($.msg) )
                              [{ $.from, area: area($p) }]
                         -> transform [{$.from, $.area}, $] ),
aggregate: fn($g,$) ( [ count($) ] )
final : fn($g,$) ($ -> transform { $g.*, n: $[0] }),
output  : { uri: “hdfs:out” } })
```


© 2009 IBM Corporation

Composite Operators



Examples:

- Join
  - Join two or more inputs on a key
  - Inner/outer/full
  - Multi-predicate, multi-way
- Merge
  - Concatenate all inputs in any order
- Union, Intersect, Difference…
- User-defined function

© 2009 IBM Corporation

CoGroup

Find users in each zipcode and the name of the city

```
$users = read("hdfs:users");
$cities = read("hbase:cities") -> expand unroll $.zips;

group $users by $zip = ($users.zip),
        $cities by $zip = ($cities.zips)
  into {$zip, n: count($users), users: $users[*].name, city: singleton($cities[*].city) };
```

Input data

<u>users</u>

```
[{ id: 12, name: "Joe Smith", zip: 94114 },
 { id: 17, name: "Ann Jones", zip: 95120 },
 { id: 19, name: "Alicia Fox", zip: 94114 } ]
```

<u>cities</u>

```
[{ city: "San Francisco", zips: [94112,94114], areas: [415] },
 { city: "San Jose", zips: [95120], areas: [408] } ]
```

after group

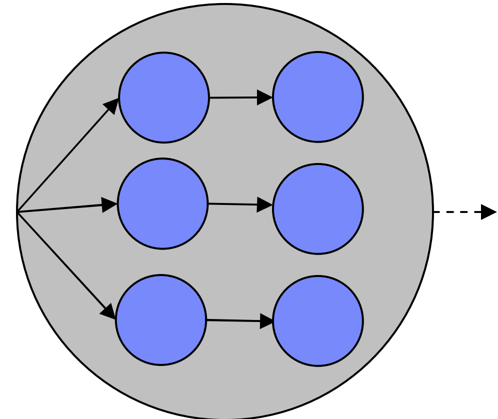
```
[{ zip: 94114,
   n: 2,
   users: [ "Joe Smith", "Alicia Fox" ],
   city: "San Francisco" },
 { zip: 95120,
   n: 1
   users: [ "Ann Jones" ],
   city: "San Jose" },
 { zip: 94112,
   n: 0,
   users: [],
   city: "San Francisco" }]
```


© 2009 IBM Corporation

Multiple Outputs: tee

Send each input item to additional output pipe

```
read("hdfs:users")
→ tee (→ filter $.zip == 94114
         → write("hdfs:zip94114") )
→ filter $.bday > date("1979-01-01")
→ write("hdfs:over30");
```



© 2009 IBM Corporation

Rough Unix analogs of Jaql

Unix	Jaql
< filename, cat	read, merge
join	join
grep	filter
cut, paste, sed, tr	transform
sort	sort
head	top
uniq	distinct
> filename	write
tee	tee

Unix: stream of bytes / lines

Jaql: stream of JSON items
more structure / types

© 2009 IBM Corporation

Other Map/Reduce Languages

- Java
  - The most basic way to write a map/reduce job in Hadoop
  - User implements a Mapper and Reducer interface and creates a JobConf
- Hadoop Streaming
  - Map and Reduce tasks implemented by external process in any language
  - Read/write data on stdin/stdout
- Pig
  - Nested relational model, but added “Map” for dynamic columns
  - From Yahoo. Hadoop subproject.
- Hive
  - Nested relational model with “Map” for dynamic columns and “Array” for lists
  - SQL-like syntax
  - From Facebook. Hadoop subproject.
- Cascading
  - A dataflow language, mostly relational.
  - Expressions are UDFs; Groovy extension
- Proprietary and non-Hadoop
  - Google Sawzall, Microsoft DryadLINQ, Microsoft Scope

© 2009 IBM Corporation

Current Roadmap

- New syntax (presented here)
  - About to be released to open source
- Robustness
  - Exception handling, …
- Features
  - Programming/scripting language integration, schema, tooling, …
- Performance
  - Query optimization, compact storage, runtime tuning, …

© 2009 IBM Corporation